Polymorphism, Subtyping, and Type Inference in MLsub

Stephen Dolan and Alan Mycroft

January 18, 2017

Computer Laboratory
University of Cambridge

The `select` function

$$\texttt{select}\ p\ v\ d = \texttt{if}\ (p\ v)\ \texttt{then}\ v\ \texttt{else}\ d$$

In ML, `select` has type scheme

$$\forall\alpha.\,(\alpha \to \texttt{bool}) \to \alpha \to \alpha \to \alpha$$

Data flow in `select`

$$\texttt{select}\ p\ v\ d = \texttt{if}\ (p\ v)\ \texttt{then}\ v\ \texttt{else}\ d$$

$$\begin{tikzcd}
 & v \arrow[dl] \arrow[dr] & & d \arrow[dl] \\
\text{argument to } p & & \text{result} &
\end{tikzcd}$$

Data flow in `select`

$$\texttt{select}\ p\ v\ d = \texttt{if}\ (p\ v)\ \texttt{then}\ v\ \texttt{else}\ d$$

$$\begin{tikzcd}
 & v \arrow[dl] \arrow[dr] & & d \arrow[dl] \\
\text{argument to } p & & \text{result} &
\end{tikzcd}$$

In MLsub, `select` has this type scheme:

$$\forall \alpha, \beta.\, (\alpha \to \texttt{bool}) \to \alpha \to \beta \to (\alpha \sqcup \beta)$$

The MLsub Type System

$$\Gamma \vdash e : \tau$$

$$\Gamma \vdash e : \tau$$

Expressions of MLsub

We have functions

$$x \qquad \lambda x.e \qquad e_1\, e_2$$

... and records

$$\{\ell_1 = e_1, \ldots, \ell_n = e_n\} \qquad e.\ell$$

... and booleans

$$\mathsf{true} \qquad \mathsf{false} \qquad \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$$

... and `let`

$$\hat{\mathbf{x}} \qquad \mathsf{let}\ \hat{\mathbf{x}} = e_1\ \mathsf{in}\ e_2$$

$$\Gamma \vdash e : \tau$$

Typing rules of MLsub

MLsub is

ML $+$

$$(\textsc{Sub}) \quad \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash e : \tau_2} \; \tau_1 \leq \tau_2$$

$$\Gamma \vdash e : \tau$$

Constructing Types

The standard definition of types looks like:

$$\tau ::= \bot \mid \tau \to \tau \mid \top$$

(ignoring records and booleans for now)

Constructing Types

The standard definition of types looks like:

$$\tau ::= \bot \mid \tau \to \tau \mid \top$$

(ignoring records and booleans for now)

with a subtyping relation like:

$$\frac{}{\bot \leq \tau} \qquad \frac{}{\tau \leq \top} \qquad \frac{\tau_1' \leq \tau_1 \quad \tau_2 \leq \tau_2'}{\tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'}$$

Lattices

These types form a lattice:

- least upper bounds $\tau_1 \sqcup \tau_2$
- greatest lower bounds $\tau_1 \sqcap \tau_2$

Lattices

These types form a lattice:

- least upper bounds $\tau_1 \sqcup \tau_2$
- greatest lower bounds $\tau_1 \sqcap \tau_2$

$$\frac{e_1 : \tau_1 \qquad e_2 : \tau_2}{\text{if } \texttt{rand ()} \text{ then } e_1 \text{ else } e_2 : \tau_1 \sqcup \tau_2}$$

Bizzarely difficult questions

Is this true, for all α?

$$\alpha \to \alpha \leq \bot \to \top$$

Bizzarely difficult questions

Is this true, for all α?

$$\alpha \to \alpha \leq \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \leq (\alpha \to \bot) \sqcup \alpha$$

Bizzarely difficult questions

Is this true, for all α?

$$\alpha \to \alpha \leq \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \leq (\alpha \to \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on α.

Bizzarely difficult questions

Is this true, for all α?

$$\alpha \to \alpha \leq \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \leq (\alpha \to \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on α.

And *only* by case analysis.

Extensibility

Let's add a new type of function $\tau_1 \overset{\circ}{\rightarrow} \tau_2$.

Extensibility

Let's add a new type of function $\tau_1 \overset{\circ}{\to} \tau_2$.

It's a supertype of $\tau_1 \to \tau_2$

"function that may have side effects"

Extensibility

Let's add a new type of function $\tau_1 \overset{\circ}{\to} \tau_2$.

It's a supertype of $\tau_1 \to \tau_2$

"function that may have side effects"

Now we have a counterexample:

$$\alpha = (\top \overset{\circ}{\to} \bot) \overset{\circ}{\to} \bot$$

Extensibility and vacuous reasoning

$$\tau_r = \text{some record type}$$
$$\tau_f = \text{some function type}$$

Is $\tau_r \sqcap \tau_f \leq \text{bool}$?

Extensibility and vacuous reasoning

$$\tau_r = \text{some record type}$$
$$\tau_f = \text{some function type}$$

Is $\tau_r \sqcap \tau_f \leq \text{bool}$?

Yes, vacuously.

Extensible type systems

Two techniques give us an extensible system:

- Add explicit type variables as indeterminates
  gets rid of case analysis

Extensible type systems

Two techniques give us an extensible system:

- Add explicit type variables as indeterminates

  gets rid of case analysis
- Require a distributive lattice

  gets rid of vacuous reasoning

Resulting types

We end up with all the standard types

Resulting types

We end up with all the standard types

... with the same subtyping order

Resulting types

We end up with all the standard types

... with the same subtyping order

... but we identify fewer of the weird types

$$\{\texttt{foo} : \mathsf{bool}\} \sqcap (\top \to \top) \not\leq \mathsf{bool}$$

Type Inference with Polar Types

Input and output types

$\tau \sqcup \tau'$: produces a value which is a τ or a τ'
$\tau \sqcap \tau'$: requires a value which is a τ and a τ'

$\sqcup$ is for outputs, and $\sqcap$ is for inputs.

Input and output types

$\tau \sqcup \tau'$: produces a value which is a τ or a τ'
$\tau \sqcap \tau'$: requires a value which is a τ and a τ'

$\sqcup$ is for outputs, and $\sqcap$ is for inputs.

Divide types into

- *output types* τ^+
- *input types* τ^-

Polar types

$$
\begin{aligned}
\tau^+ &::= \mathsf{bool} \mid \tau_1^- \to \tau_2^+ \mid \{\ell_1 : \tau_1^+, \ldots, \ell_n : \tau_n^+\} \mid \\
&\quad\ \alpha \mid \tau_1^+ \sqcup \tau_2^+ \mid \bot \mid \mu\alpha.\tau^+ \\
\tau^- &::= \mathsf{bool} \mid \tau_1^+ \to \tau_2^- \mid \{\ell_1 : \tau_1^-, \ldots, \ell_n : \tau_n^-\} \mid \\
&\quad\ \alpha \mid \tau_1^- \sqcap \tau_2^- \mid \top \mid \mu\alpha.\tau^-
\end{aligned}
$$

Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types
- Unifying two output types
- Using the output of one expression as input to another

Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types

  Introduce $\sqcap$

- Unifying two output types

  Introduce $\sqcup$

- Using the output of one expression as input to another

  $\tau^+ \leq \tau^-$ *constraint*

Eliminating variables, ML-style

Suppose we have an identity function

$$\alpha \to \alpha$$

Eliminating variables, ML-style

Suppose we have an identity function, which uses its argument as a τ

$$\alpha \to \alpha \mid \alpha = \tau$$

Eliminating variables, ML-style

Suppose we have an identity function, which uses its argument as a τ

$$\begin{aligned} & \alpha \to \alpha \mid \alpha = \tau \\ \equiv^{\forall}\ & \tau \to \tau \end{aligned}$$

Eliminating variables, ML-style

Suppose we have an identity function, which uses its argument as a τ

$$
\begin{aligned}
& \alpha \to \alpha \mid \alpha = \tau \\
\equiv^{\forall}\ & \tau \to \tau
\end{aligned}
$$

The substitution $[\tau/\alpha]$ solves the constraint $\alpha = \tau$

Eliminating variables, MLsub-style

Suppose we have an identity function, which uses its argument as a τ^-.

$$\alpha \to \alpha \mid \alpha \leq \tau^-$$

Eliminating variables, MLsub-style

Suppose we have an identity function, which uses its argument as a τ^-.

$$\begin{aligned} &\alpha \to \alpha \mid \alpha \leq \tau^- \\ \equiv^\forall\ &(\alpha \sqcap \tau^-) \to \alpha \end{aligned}$$

Eliminating variables, MLsub-style

Suppose we have an identity function, which uses its argument as a τ^-.

$$
\begin{aligned}
&\alpha \to \alpha \mid \alpha \leq \tau^- \\
\equiv^\forall\ &(\alpha \sqcap \tau^-) \to \alpha
\end{aligned}
$$

The *bisubstitution* $[\alpha \sqcap \tau^- / \alpha^-]$ solves $\alpha \leq \tau^-$

Decomposing constraints

We only need to decompose constraints of the form $\tau^+ \leq \tau^-$.

$$
\begin{aligned}
\tau_1 \sqcup \tau_2 \leq \tau_3 \quad &\equiv \quad \tau_1 \leq \tau_3,\ \tau_2 \leq \tau_3 \\
\tau_1 \leq \tau_2 \sqcap \tau_3 \quad &\equiv \quad \tau_1 \leq \tau_2,\ \tau_1 \leq \tau_3
\end{aligned}
$$

Thanks to the input/output type distinction, the hard cases of $\tau_1 \sqcap \tau_2 \leq \tau_3$ and $\tau_1 \leq \tau_2 \sqcup \tau_3$ can never come up.

Combining solutions

We solve a system of multiple constraints C_1, C_2 by:

- Solving C_1, giving a bisubstitution ξ
- Applying that to C_2
- Solving ξC_2, giving a bisubstitution ζ

Then $\xi \circ \zeta$ solves the system C_1, C_2.

Summary

MLsub infers types by walking the syntax of the program, but must deal with subtyping constraints rather than just equalities. Thanks to:

- algebraically well-behaved types
- polar types, restricting occurrences of $\sqcup$ and $\sqcap$
- the biunification algorithm

we can always handle these constraints, producing a principal type.

Future work

```
{-# LANGUAGE ??  #-}
```

- `RankNTypes?`
- `GADTs?`
- `TypeFamilies?`
- ...

Questions?

`http://www.cl.cam.ac.uk/~sd601/mlsub`

`stephen.dolan@cl.cam.ac.uk`

Mutable references

References are generally considered "invariant".

Instead, consider `ref` a two-argument constructor

$$(\alpha, \beta)\ \texttt{ref}$$

with operations:

$$\begin{aligned} \texttt{make} &: (\alpha, \alpha)\ \texttt{ref} \\ \texttt{get} &: (\bot, \beta)\ \texttt{ref} \to \beta \\ \texttt{set} &: (\alpha, \top)\ \texttt{ref} \to \alpha \to \texttt{unit} \end{aligned}$$